MC030105

IB04/50024

Europäisches Patentamt
European Patent Office
Office européen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

03075239.8

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 03075239.8
Demande no:

Anmeldetag:
Date of filing: 24.01.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

Koninklijke Philips Electronics N.V.
Groenewoudseweg 1
5621 BA Eindhoven
PAYS-BAS

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Dataprocessing system

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F9/00

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL
PT SE SI SK TR LI

Dataprocessing system.

The invention relates to a dataprocessing system comprising a first and second processing module which are mutually coupled by a synchronizing buffer.

Two developments undermine the role of globally clocked VLSI circuits. In the first place, the trend towards system-on-chip designs leads to chips containing several processing modules which all have different cycle times. Secondly, in future technologies it will become increasingly difficult to distribute high-speed low-skew clock signals. Therefore, future chips will contain several locally clocked processing modules, which communicate through dedicated glue logic in the form of a synchronizing buffer. These heterogeneous systems are called GALS (Globally Asynchronous, Locally Synchronous) systems.

Such a synchronizing buffer is known from J. N. Seizovic. Pipeline synchronization. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 87--96, Nov. 1994. In the dataprocessing shown therein the pipeline comprises a synchronizer between each two pipeline elements. A synchronizer is disclosed having a relatively complicated structure, which comprises an RS flip-flop having a set input controlled by a first mutually exclusive (ME) gate, and a reset input controlled by a second ME-gate. Each of the ME-gates is synchronized with a clocksignal and receives a respective request signal from a logic circuit in response to a request signal from a preceding pipeline element, a request signal from a succeeding element and an internally feedback signal.

It is a purpose of the invention to provide an improved dataprocessing system which allows synchronization by relatively simple synchronization circuits. In accordance therewith the dataprocessing of the invention is defined by claim 1.

In the dataprocessing system according to the invention the pipeline elements mutually control the dataflow via a two phase handshake signal. Internally however a four phase handshake signal is used to control the latches that latch the dataflow. As will be shown in the sequel, the pipeline elements in the dataprocessing system according to the invention can be synchronized by very simple synchronization elements. In the following reference will be made to the following Annex: "Bridging Clock Domains by synchronizing the mice in the mousetrap", Joep Kessels, Ad Peeters and Suk-Jin Kim.

It is noted that the pipeline elements are known as such from WO 02/35346. However it is not disclosed herein to combine the pipeline elements with synchronizers. On the contrary it is suggested that it is essential that the pipeline is asynchronous to enable to interface with environments operating at different rates.

In an embodiment the synchronization element synchronizes the control signal with a phase of the clock signal. This can be realized with a two-phase wait component which can be constructed from a four-phase wait component as shown in Figure 5 of the Annex. The two-phase wait component comprises a comparison element, such as an XOR-gate, a four-phase wait component and a latch, wherein the latch receives a request signal in order to provide the latched request signal as an acknowledge signal. The request signal and the acknowledge signal are compared by the comparison element which provides an input signal for the four-phase wait component, which synchronizes this signal with the clock signal to provide the control signal for the latch.

In a preferred embodiment the synchronization element synchronizes the control signal with a transition (edge) of the clock signal. A four phase edge synchronizer can be obtained from a combination of two four-phase wait components as is shown in Figure 8a of the Annex. Therein the

first four-phase wait component synchronizes the input control signal with the inverted clock signal and provides the phase synchronized signal to the second four-phase wait component. The latter synchronizes the so obtained signal for a second time with the original clocksignal.

A further improvement is obtained by replacing the two wait components in Figure 6.a (or Figure 7.a) of the Annex by one up-edge component in between the two pipeline elements. An up-edge two-phase (edge) transition synchronization element is built by replacing the WAIT4 components shown in Figure 8.a of the Annex by four-phase up-edge components .

Synchronizing on edges instead of phases improves the speed of the synchronizing buffer.

Depending on whether the pipeline has to synchronize with a writing processing module, or with a reading processing module or both, several options are possible.

Synchronization with a writing processing module can be obtained in a dataprocessing system wherein the synchronization element synchronizes the second control signal (Rack) with the clock signal.

The probability of synchronization failures can be decreased further by introducing synchronization elements between subsequent pipeline elements, or in other words in dataprocessing system, wherein the pipeline comprises at least a first and a second pipeline element and wherein the synchronization element provides the second control signal (Rack) for the first pipeline element by synchronizing the latched first control signal (Rreq) of the second pipeline element with a clock signal provided by the first processing module (Wclk). This is shown in more detail in Figures 6 and 9 of the Annex, for phase synchronizing and edge synchronizing respectively.

Synchronization with a reading processing module can be obtained in a dataprocessing system, wherein the synchronization element synchronizes the first control signal (Wreq) with the clock signal.

Likewise the probability of synchronization failures can be decreased further by introducing synchronization elements between subsequent pipeline elements, i.e. in a dataprocessing system, wherein the pipeline comprises at least a first and a second pipeline element and wherein the synchronization element provides the first control signal (Wreq) for the second pipeline element by synchronizing the latched first control signal (Rreq) of the first pipeline element with a clock signal provided by the second processing module (Rclk). This is shown in more detail in Figures 7 and 10 of the Annex, for phase synchronizing and edge synchronizing respectively.

Synchronizing with a reading processing module on one end of the pipeline and with a writing processing module on the other end of the pipeline is possible, by composing the pipeline of a write section, a read section and an intermediate section as is described in more detail in paragraph 4 of the Annex.

Claim 8 defines a very favorable embodiment of the dataprocessing system according to the invention. Therein the controller further comprises a further comparator for comparing the first control signal (Wreq) with the latched first control signal (Rreq), the controller being arranged for providing the latch control signal (d) in response to first control signal (Wreq), the latched first control signal (Rreq) and the second control signal (Rack), and wherein the synchronization element provides the latch control signal (e) to the latches by synchronizing the latch control signal (d) provided by the comparator with the clock signal (Clk). Such an embodiment is shown in Figure 11 of the Annex. In that embodiment the synchronization element is incorporated into the pipeline element, and the so obtained pipeline element is universally applicable in each section of the pipeline. Additionally the pipeline element according to claim 8 allows a further improvement in clock frequency for the write side as is shown in Table 1 of the Annex.

It has been recognized by the inventors that it is not necessary to synchronize both the transfer of the full buckets and the empty buckets in the pipeline. This implies that the further comparator can be dispensed with. Consequently the comparator comparing the signals Rack and Rreq can directly provide its output signal to the synchronization element. The AND-gate in Figure

11 therewith is superfluous. Instead of using a edge synchronization in Figure 11 the control signal for the latch may be controlled by phase synchronization.

Figure 1 schematically shows a GALS system comprising a first and second mutually asynchronous processor modules 10, 20 which are coupled to each other by a synchronizing buffer 30. The buffer with an input side W (for Write) coupled to the first processor module 10 and an output side R (for Read) coupled to the second processor module 20. Each side has a clocked interface with an independent clock (Wclk and Rclk). All other input/output signals are valid at the rising edge of the corresponding clock signal. The clocked protocol at each buffer interface is not a handshake protocol, but a more appropriate symmetric rendez-vous protocol in which both the buffer and the environment indicate their readiness to perform a transfer operation by making a dedicated signal high.

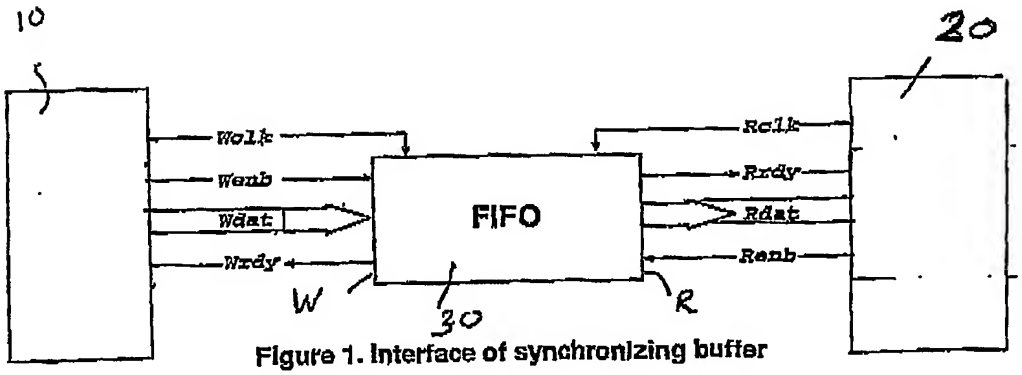


Figure 1. Interface of synchronizing buffer

Bridging Clock Domains by synchronizing the mice in the mousetrap

Joep Kessels,* Ad Peeters,* and Suk-Jin Kim§

Abstract

We present the design of a first-in first-out buffer that can be used to bridge clock domains in GALS (Globally Asynchronous, Locally Synchronous) systems. Both the input and output side of the buffer have an independently clocked interface. The design of these kind of buffers inherently poses the problems of metastability and synchronization failure. In the proposed design the probability of synchronization failure can be decreased exponentially by increasing the buffer size. Consequently, at system level one can trade off between safety and low latency. The design is based on two well-known ideas: pipeline synchronization and mousetrap buffers. We first combine both ideas and then in two steps improve the design.

Keywords: *GALS systems, data synchronizer, pipeline synchronization, mousetrap buffer, bridging clock domains.*

1. Introduction

Two developments undermine the role of globally clocked VLSI circuits. In the first place, the trend towards system-on-chip designs leads to chips containing several IP modules which all have different cycle times. Secondly, in future technologies it will become increasingly difficult to distribute high-speed low-skew clock signals. Therefore, future chips will contain several locally clocked submodules, which communicate through dedicated glue logic. These heterogeneous systems are called GALS (Globally Asynchronous, Locally Synchronous) systems [1]. Two kinds of GALS systems can be distinguished depending on the way the synchronous submodules communicate.

- In a *clock synchronization* system, the submodules have so-called pausible clocks, which are ring oscillators that can be halted. Safe communication is obtained by synchronizing the clocks [9, 10, 14, 8, 6, 3].
- In a *data synchronization* system, the submodules have free-running clocks and the data being communicated from one clock domain to the other is synchronized. A simple solution is the well-known two-register or double-latch synchronizer [9, 5]. More elaborate synchronizing schemes are based on first-in first-out buffers. The solution presented in [2] uses (distributed) pointer-based buffers offering both low latency and low power dissipation. However, since reading and writing is done via a bus connecting all cells, it will be difficult to obtain a high throughput. This holds in particular when large distances have to be bridged. In [11] a solution is presented based on ripple buffers. Compared to pointer-based buffers, ripple buffers have longer latencies and dissipate more power. However, they allow distributed placements with short point-to-point interconnects. Therefore, ripple buffers can offer higher throughputs.

Data synchronization systems inherently have to deal with metastable states, which have to be resolved within a given time period. When choosing this period one has to trade off between safety and low latency. Clock synchronization systems are safer in that they wait until the metastable states have been resolved. Moreover, a large subset of clock synchronization systems can be designed without any arbitration [9, 10, 1, 6]. Despite this technical advantage of clock synchronization systems, synchronous designers tend to prefer the more familiar data synchronization systems.

We present a pipeline synchronizer based on the mousetrap buffer [12]. The paper is organized as follows. Section 2 gives the specification of the synchronizing buffer and in section 3 we introduce and analyse the design of the mousetrap buffer. In section 4 we apply pipeline synchronization in the mousetrap buffer. We first combine both ideas and then in two steps improve the design. In section 5 we summarize the differences between the three designs. All the designs that we discuss have been implemented with a 16-

*Philips Research Laboratories, NL-5656 AA Eindhoven, The Netherlands, {joep.kessels,ad.peeters}@philips.com
§Kwang-Jn Institute of Science and Technology, Kwang-ju, 500-712 South Korea, sjkim@kjist.ac.kr

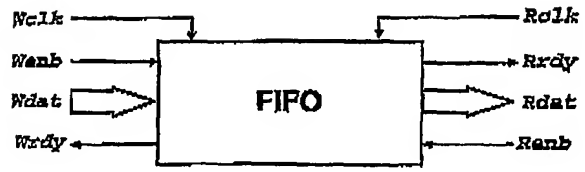


Figure 1. Interface of synchronizing buffer

bit wide data path in a 0.18 μm CMOS technology. They have been simulated using back annotation with estimated wire loads.

2. Specification of the synchronizing buffer

Fig. 1 shows a buffer with an input side *W* (for Write) and an output side *R* (for Read). Each side has a clocked interface with an independent clock (*Wclk* and *Rclk*). All other input/output signals are valid at the rising edge of the corresponding clock signal. The clocked protocol at each buffer interface is not a handshake protocol, but a more appropriate symmetric *rendez-vous* protocol in which both the buffer and the environment indicate their readiness to perform a transfer operation by making a dedicated signal high. The buffer signal is called *rdy* (for ready) and the signal from the environment is called *enb* (for enable). A data transfer only occurs if both the ready and enable signal are high at a rising clock edge. On the input side *Wdat* must be valid when *Wenb* is high and on the output side *Rdat* is valid when *Rrdy* is high. Note that the specification of the interface allows two buffers to be connected directly, provided the two interconnected interfaces share the same clock.

3. The mousetrap

The design of the buffer is based on the well-known mousetrap buffer [12], which is a ripple buffer using 2-phase single-rail handshake signalling for the communication between two neighbouring cells. The mousetrap buffer has several properties that make it very attractive for clock bridging in GALS systems.

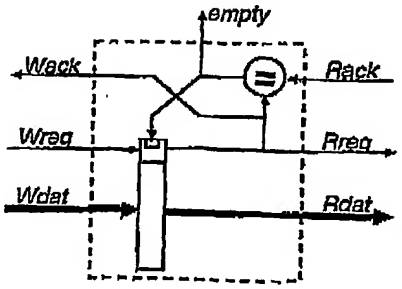
- In many GALS systems clock bridging also implies bridging distances, in which case the transmission delays are important. For this reason a two-phase protocol is much more attractive than a four-phase protocol.
- The mousetrap buffer cell has a short cycle time, which means that it allows high throughputs (fast clocks) in the synchronizing buffers. Moreover, for a given clock frequency, a shorter cycle time implies a smaller probability of synchronization failure.
- An empty mousetrap buffer offers a minimum latency of only one latch delay per cell.
- All cells can be filled, which means that the buffer capacity is equal to the number of cells.
- Since the design does not contain any special asynchronous elements (such as C-elements), it is more easily understood and accepted by conventional synchronous designers, which are often the designers applying the GALS interfacing circuitry.

3.1. One cell

Fig. 2(a) shows the design of a mousetrap cell (MT) and Fig. 2(b) gives some invariants that hold when the cell is in a quiescent state. First we note that (read request) signal *Rreq* is equal to (write acknowledge) signal *Wack* (invariant 1). Moreover, signal *empty* is high if and only if the read handshake signals are equal (invariant 2). Signal *empty* controls a register consisting of a control bit and a set of data bits. If signal *empty* is high, all latches in the register are transparent, which means that the outputs of the latches are equal to the inputs (invariant 3). From these invariants it follows that if signal *empty* is high (quiescent state then means waiting for an input), all four handshake signals are equal.

Fig. 2(c) describes the behaviour of the cell for which we use the convention introduced in [7]. All cells start by initializing the control latch with outgoing signal *Rreq* to the same value, say false, which implies that initially all handshake signals are equal and, hence, all cells are empty. Subsequently the cell executes an endless loop ('$A; B$' means sequential execution of A and B and 'A^*' means infinite execution of A). In each step of the loop the cell first waits until the read handshake signals are equal ('$[C]$' means wait until C holds). When this happens, signal *empty* becomes high and the latches in the register become transparent. The cell now waits until signal *Wreq* differs from signal *Rreq* (from invariant (1) it follows *Wreq* and *Wack* then differ), which means that its write neighbour is full (offering data). Subsequently the transparent latches take over the input signals ('$A \parallel B$' means concurrent execution of A and B). Consequently, the write handshake signals become equal (write neighbour becomes empty), and signal *Rreq* is inverted (cell becomes full). Signal *empty* then goes low making the latches opaque and the cell executes a next loop step. Note that passing a data item (full bucket) is done via the request signals, whereas the acknowledge signals are used to return an empty bucket.

Let us consider the feed-back loop circuit consisting of the equal gate and the control latch, which has two input signals *Wreq* and *Rack* and two output signals *Wack* and *empty*. If we ignore output signal *empty*, the circuit behaves as follows



(a) Design

$$Rreq = Wack \tag{1}$$
$$empty \Leftrightarrow (Rreq = Rack) \tag{2}$$
$$empty \Rightarrow (Rreq = Wreq) * (Rdat = Wdat) \tag{3}$$

(b) Invariants

$Rreq := false;$
$([Rreq = Rack];\ empty\uparrow$
$; [Wreq \neq Wack]$
$; (Rdat := Wdat \parallel Wack := Wreq \parallel Rreq := -Rack)$
$;\ empty\downarrow$
$)^*$

(c) Behaviour

Figure 2. Free-running mousetrap cell (MT)

$$([(Rreq = Rack) * (Wreq \neq Rreq)];\ Rreq := Wreq)^*,$$

which is equal to

$$([(Wreq \neq Rack) * (Wreq \neq Rreq)];\ Rreq := Wreq)^*.$$

The condition that the two request signals must differ is superfluous, since it only prevents idle operations. Therefore the behaviour can be rewritten as

$$([Wreq \neq Rack];\ Rreq := Wreq)^*,$$

which is the behaviour of a symmetric C-element with an inverter at the *Rack* input. In this respect the design corresponds with the micropipeline design presented in [13].

The circuit offers, however, more functionality. Additional output signal *empty* can be used to control the data latches in a clever way: it gives automatic delay matching (provided the enable signals are skew-free) and during each two-phase handshake signal *empty* makes a complete latch control cycle from transparent to opaque and back to transparent again. Therefore a mousetrap buffer is only full if all cells are full.

The C-element in the mousetrap has one disadvantage: it contains an extended isochronic fork, since the output changes before the latches have become opaque (state is settled). Signal *Wack* indicates that the write data will be latched after a certain time (XNOR delay plus the hold time of a latch). Therefore, signal *Wack* comes with a timing constraint: signal *Wreq* should be stable until the latch is closed. If the write neighbour is also a mousetrap cell, this requirement is automatically fulfilled (we come back to the delay asymmetries in section 3.3). Note that transmission delays between the cells help to make the design more robust.

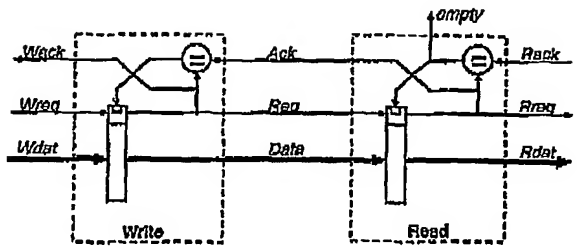
3.2. Two communicating cells

Fig. 3 shows two communicating mousetrap cells: a write and a read cell. We define the *cycle time* of the buffer as the minimum time it takes for two cells to communicate one symbol from the write cell to the read cell. From Fig. 3(a) it follows that the cycle time of the mousetrap cell is δ(XNOR)+δ(LATCH) [1] in the write cell and only δ(LATCH) in the read cell. In our technology the total delay adds up to 0.86 nsec leading to a maximum throughput of 1.26 Gsymbols/sec.

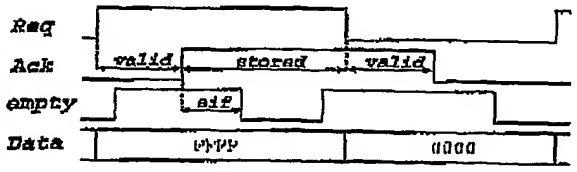
Fig. 3(b) and Fig. 3(c) show some simulation results when the two-place buffer runs at full speed. Fig. 3(b) shows the internal communications. The write cell indicates that the data signals are valid by changing signal *Req* (thereby making the handshake signals different); the read cell indicates that the data signals will be stored by changing signal *Ack* (thereby making the handshake signals equal again). The cycle time only includes the delays needed to pass on the handshake signals (rising edge delay of the XNOR-gate and latch delay). The events needed to close the latch are performed concurrently with the cycle activity of the read cell. The figure shows that the latches in the read cell are still transparent (signal *empty* is high) when the cell has signalled that it has stored the data (period *eif*). This is due to the extended isochronic fork in the design and period *eif* corresponds to an XNOR delay. Note that signal *empty* goes high at about the same time when signal *Req* makes a transition. From this fact it follows that the buffer runs at full speed (full bucket from the write cell and empty bucket in the read cell arrive at about the same time).

Fig. 3(c) shows the external communications. Note the phase shift between the input and output operations, which will be discussed in the next section.

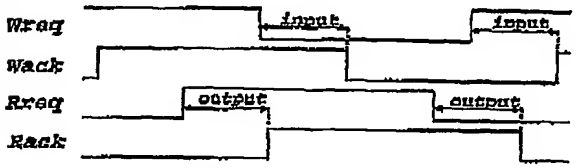
[1] $\delta(A)$ stands for the delay of gate A



(a) Design



(b) Timing simulation internal communications



(c) Timing simulation external communications

Figure 3. Two communicating mousetrap cells

3.3. About asymmetries in delay

Several asymmetries play a role with respect to timing.

- The inputs of a standard cell typically have different delays. In our design this holds for the XNOR-gate and the latch (delay from enable to data-out differs from the delay from data-in to data-out). The asymmetry for the XNOR-gate can be used to reduce the timing requirements that follow from the extended isochronic fork by connecting the request signal to the faster input and the acknowledge signal to the slower one.
- The delays for rising and falling edges differ for all components. Therefore if the buffer runs at full speed, the high and the low periods of the handshake signals differ, which for a two-phase protocol implies that two consecutive communications take different times. Note that in Fig. 3(b) a rising edge in signal *Req* occurs before signal *empty* goes high, whereas for falling edges in signal *Req* it is the other way around.
- The most important asymmetry, however, is the difference between the times it takes to pass a full or an empty bucket. Passing a full bucket (δ(LATCH)) is faster than passing an empty bucket (δ(XNOR)+δ(LATCH)). Therefore the data *valid* period is shorter than the *stored* period. It also leads to a phase shift in the external handshakes of this two-place buffer of which *Wreq* and *Wack* are the input handshake signals and *Rreq* and *Rack* are the output handshake signals (see Fig. 3(c)). In a perfectly symmetric buffer the input and output handshakes would occur exactly at the same time (without any phase shift), which means that such a two-place buffer running at full speed would always contain one data item. However, a mousetrap buffer with $2*N$ cells running at full speed contains less than N data items. The asymmetry is not relevant for the maximum throughput of a free-running buffer, it only leads to a phase shift between the input and output operations. We will see, however, that in a synchronizing buffer (buffer synchronized by a clock signal) the asymmetry has consequences for the maximum throughput.

  Due to the asymmetry a free-running buffer containing N data items and closed in a loop, will not run at full speed. A similar kind of asymmetry has also been observed in [4].

4. Pipeline synchronization

A pipeline-synchronization buffer consists of three sections: a write section, an intermediate section and a read section. The write section synchronizes the input operations with the write clock, while the read section synchronizes the output operations with the read clock. The intermediate section is an asynchronous buffer which serves to decouple the two synchronizing sections. The design of all three sections is based on ripple buffers. The transformation from a ripple buffer into a synchronizing buffer is presented in [11]. This transformation is based on inserting in between two neigbouring cells a component that synchronizes the handshakes with a clock.

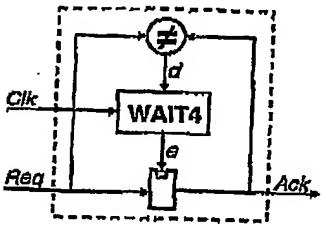
We define the *maximum clock frequency* of a synchronizing section as the maximum clock frequency at which that section can transfer one data item every clock cycle.

(a) Design

([Req ∗ Clk]; Ack↑
; [Req]; Ack↓
)*

(b) Behaviour

Figure 4. Four-phase wait component (WAIT4)



(a) Design

([Ack ≠ Req]; d↑
; [Clk]; e↑
; Ack := Req; d↓
; e↓
)*

(b) Total behaviour

$([(Ack \neq Req)]; [Clk]; Ack := Req)^{\omega}$

(c) External behaviour

Figure 5. Two-phase wait component (WAIT2)

4.1. Synchronization based on clock phases

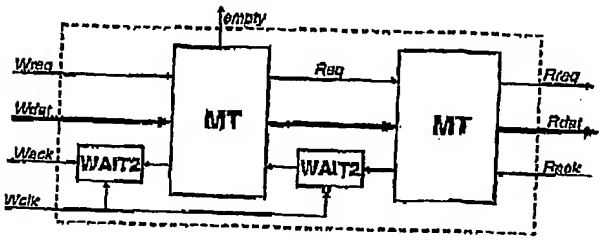
The synchronizing components in the design presented in [11] are so-called WAIT-components which synchronize handshakes with a clock phase. The basic WAIT-component is a four phase wait component (called WAIT4), which delays the completion of a four-phase handshake until an additional input signal *Clk* is high. Since signal *Clk* can go low when the handshake starts, a conflict can occur implying that an arbiter is needed in the design of the component. Fig. 4 shows the design of the WAIT4-component based on a basic arbiter (also called mutex). As long as *Clk* is low, the arbiter grants the (virtual) right to proceed to the upper request implying that handshakes are paused until *Clk* is high.

However, since the mousetrap-buffer uses two-phase handshake signalling, we need a two-phase WAIT-component (called WAIT2). Fig. 5 shows the design such a WAIT2-component based on a WAIT4-component. If the two handshake signals *Req* and *Ack* differ, signal *d* goes high after which the WAIT4-component waits until signal *Clk* is high before it makes signal *e* high. Subsequently, the latch becomes transparent and signal *Ack* becomes equal to *Req*. When the handshake signals are equal, the signals *d* and *e* become low again.

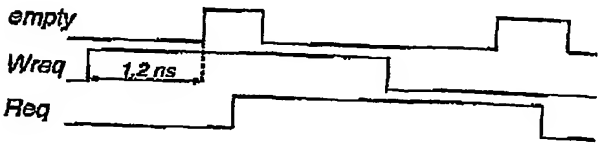
We define the 4TO2-circuit as the additional circuitry needed to convert a WAIT4 into a WAIT2-component. Since this circuit is very similar to the control circuit of a mousetrap cell, it has a similar timing constraint. This design of a WAIT2-component is different from the one presented in [11], which uses two WAIT4-components.

We now give the designs of *phase-synchronizing buffers*, which are buffers that synchronize handshakes with the phases of a clock. For both the write and the read section we present a two-place phase-synchronizing buffer based on WAIT2-components.

Fig. 6(a) shows the design of the write synchronizing buffer, which synchronizes the transfer of empty places. Since empty places are transferred by making the acknowledge signal equal to the request signal, the acknowledge signal is synchronized with the write clock. Fig. 6(b) shows a timing simulation when the buffer runs at ful speed. We see that full buckets (transitions in signal *Wreq*) arrive before the corresponding empty buckets (rising edges in signal

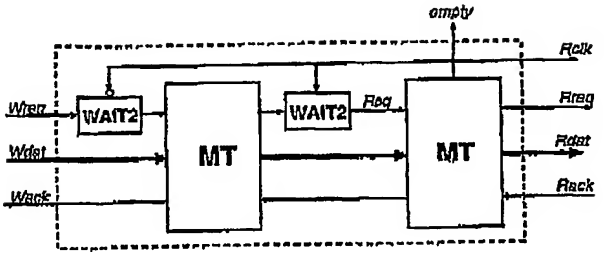


(a) Design

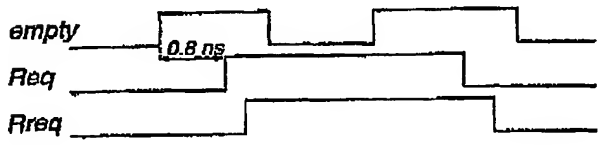


(b) Timing simulation

Figure 6. Write phase-synchronizing buffer with two mouse-trap cells



(a) Design



(b) Timing simulation

Figure 7. Read phase-synchronizing buffer with two mouse-trap cells

empty).

Fig. 7(a) shows the design of the read synchronizing buffer, which synchronizes the transfer of data items. Since data items are transferred by changing the request signal, this signal is synchronized with the clock. Fig. 7(b) shows that in this buffer, empty buckets (rising edges in signal *empty*) arrive before the corresponding full buckets (transitions in signal *Wreq*).

Both buffers, when running freely (wait components always enabled), offer a maximum throughput of about 570 Msymbols/sec. We now come back to the asymmetry in time it takes to pass a full or an empty bucket. A synchronizing buffer running at full speed transfers one data item every clock cycle, which implies that the input and output operations of a two-place buffer occur simultaneously synchronized by the clock. Therefore the phase shift that allowed each cell in a free-running buffer to operate at full speed, does not exist in a synchronizing buffer. Consequently, in a synchronizing buffer the asymmetry leads to a decrease in the maximum throughput: the larger the asymmetry, the larger the performance loss.

We saw that in the free-running buffer the acknowledge path is δ(XNOR) slower than the request path. In the design of the write synchronizing buffer we have inserted a WAIT2-component in the acknowledge path leading to a delay difference of δ(XNOR)+δ(WAIT2). Consequently, the full buckets arrive 1.2 ns before the empty ones resulting in a maximum clock frequency of 316 MHz. In the design of the read synchronizing buffer, however, we have inserted a WAIT2 component in the request path leading to a delay difference of $max(\delta(\text{XNOR}),\delta(\text{WAIT2}))$. Since δ(WAIT2) is larger than δ(XNOR), the empty buckets arrive before the full ones. The time difference is only 0.8 ns resulting in a maximum clock frequency of 403 MHz.

4.2. Synchronization based on clock edges

The two-place synchronizing buffer based on WAIT-components can be transformed into a faster one by bringing the two WAIT2-components together (for instance in between the mousetrap cells). The two combined WAIT2-components form a so-called EDGE-component.

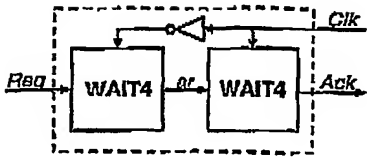
Fig. 8(a) shows the UE4-component that synchronizes the upgoing phase of a four-phase handshake with the rising edges of a clock. The component is constructed by connecting two WAIT4-components. Fig. 8(b) describes the total behaviour of the UE4-component, which consists of the parallel composition of the beaviour of two WAIT4-components. The first WAIT4-component waits until both the request signal is high and the clock signal is low and then makes intermediate signal *ar* high. The second WAIT4-component waits until both signal *ar* and the clock are high and then makes signal *Ack* high. When the request signal goes low, signal *ar* and, consequently, signal *Ack* go low as well. At falling clock edges the inverter delay between the clock signals of the two WAIT-components takes care of closing the second WAIT-component before the first one is opened. If we abstract from the internal behaviour we get the external behaviour which is shown in Fig. 8(c).

To obtain a two-phase version of the EDGE-component (UE2) we can use the 4TO2-circuit again.

Two-place edge-synchronizing buffers can be obtained by connecting two free running buffer cells and inserting an UE2-component in one of the interconnecting handshake signals: in the acknowledge signal for the write buffer and in the request signal for the read buffer. Compared to the phase-synchronizing buffers presented in the previous section, we now have reduced the number of 4TO2-circuits in a two-place buffer from two to one. Therefore, edge-synchronizing buffers are faster than phase-synchronizing ones. The maximum frequency is for the write buffer 478 MHz and for the read buffer 588 MHz.

An edge-synchronizing buffer offers the following advantages when compared to a phase-synchronizing one:

- it offers a higher maximum clock frequency;
- for the same clock frequency it gives a smaller probability of synchronization failure (since the circuit overhead is less);



(a) Design

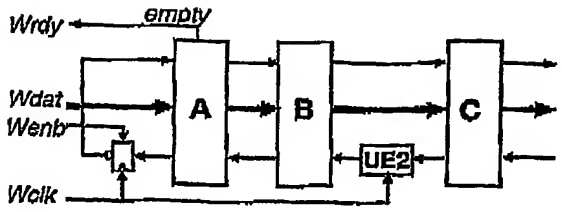
$([Req * \overline{Clk}];\ ar\uparrow;\ [\overline{Req}];\ ar\downarrow)^* \ \|$
$([ar * Clk];\ Ack\uparrow;\ [\overline{ar}];\ Ack\downarrow)^*$

(b) Total behaviour

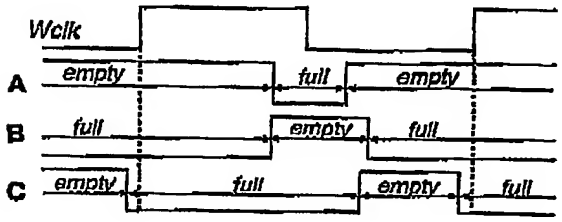
$([Req * \overline{Clk}];\ [Clk];\ Ack\uparrow;\ [\overline{Req}];\ Ack\downarrow)^*$

(c) External behaviour

Figure 8. Four-phase up-edge component (UE4)



(a) Design



(b) Timing simulation

Figure 9. Edge-synchronizing write section

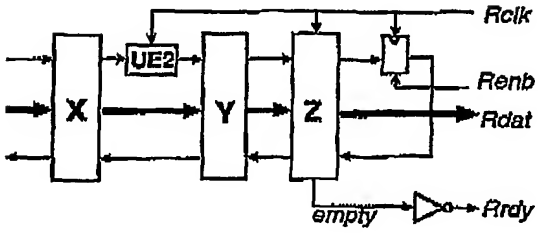
- the timing of the external events in relation to rising clock edges is well-defined (which is very important when we add the circuitry offering the clocked interface);
- it has a behaviour that is independent of the ratio between the high and the low phase of the clock signal.

4.3. Total buffer design

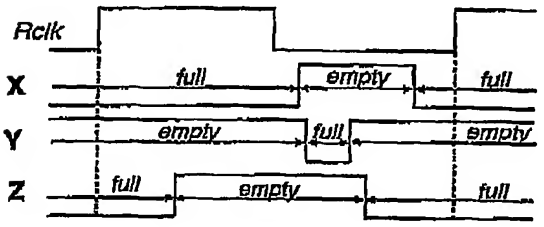
Fig. 9(a) shows the write section with its clocked interface. The design is based on edge synchronization and it contains three mousetrap cells: A, B and C. The write side of cell A is connected to circuitry offering the clocked write interface to the environment. The flipflop in the design has a clock enable signal *Wenb*. When the cell is empty, *Wrdy* is high and the write handshake signals are equal. The cell then waits for a rising edge of *Wclk* with *Wenb* high and when this happens it writes data in empty cell A by making the write request signal the inverse of the write acknowledge signal. The output signals of cell A must meet the setup and hold requirements with respect to clock signal *Wclk*. This holds not only for signal *Wrdy* but also for the write acknowledge signal. The timing of these external events is better predictable for an edge-synchronizing buffer than for a phase-synchronizing one.

Clocked input cell A is followed by the cells B and C, which form a two-place edge-synchronizing buffer. This buffer synchronizes input operations with the write clock. Fig. 9(b) shows the *empty* signals of the three cells when the buffer operates at full speed. Just before the rising clock edge, we have the following situation: cell A is empty, and both cell B and C are full containing the same data item (the acknowledge signal making cell B empty is being held up until the rising clock edge occurs). After the rising edge cell A becomes full, cell B becomes empty and cell C remains full. Subsequently, the asynchronous communication within the cell pairs (A and B) and (C and its read neighbour) restores the initial state before the next rising clock edge. Note that these cell pairs can be seen as master/slave pairs, which when operating at full speed continuously contain one data item. At every rising clock edge the master receives data and then this data is passed on asynchronously to the slave. However, when the buffer is full both the master and the slave contain a data item. Note also that when the complete buffer section is empty, the latches in all three cells are transparent. Therefore in that situation this buffer section has a very small latency of only three latch delays.

A similar analysis holds for the read section shown in Fig. 10(a) which contains the mousetrap cells X, Y and Z. Cell Z starts in the empty state, which implies *Rrdy* low



(a) Design



(b) Timing simulation

Figure 10. Edge-synchronizing read section

and its read handshake signals equal. When *Z* becomes full, *Rrdy* goes high and the read request signal becomes different from the read acknowledge signal. As soon as now a rising edge of *Rclk* occurs, the read handshake signal become equal which brings the cell back to its initial state with *Rrdy* low. Fig. 10(b) shows the *empty* signals of the three cells when the buffer operates at full speed. Just before the rising clock edge, cell *X* is full, cell *Y* is empty and cell *Z* is full. After the rising edge cell *X* becomes empty, cell *Y* becomes full and cell *Z* becomes empty. Subsequently, the asynchronous communication within the cell pairs (*X* and and its write neighbour) and (*Y* and *Z*) restores the initial state before the next rising clock edge. Note that the latency of this section is one clock tick.

By adding a two-place synchronizing buffer to one of the synchronizing sections, one can asymptotically decrease the probability of synchronization failure at the cost of increasing the latency.

4.4. Universal synchronization cell

The increase in performance that was obtained by going from clock-phase to clock-edge synchronization came from the fact that for every pair of buffer cells we could reduce the number of 4TO2-circuits from two to one. When comparing the 4TO2-circuit with the synchronizing circuit of a mousetrap cell one can notice a strong similarity. It is therefore tempting to integrate the synchronizing circuit in the mousetrap cell.

Fig. 11(a) shows the design of such an integrated cell. Since in our technology the delay of the combinational circuitry consisting of the XNOR, XOR and AND-gate (which can largely be simplified) is comparable to the delay of a single XNOR-gate, we have done away with the overhead of the last 4TO2-ciruit. With respect to performance, the design has an additional advantage: by doing the clock synchronization on a signal that is common for the transfer of both full and empty buckets, the asymmetry between the two transfer paths has become very small.

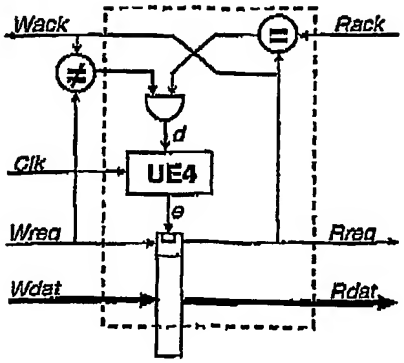
Fig. 11(b) describes the behaviour of the cell. The cell –when empty– waits until its write neighbour is full and then at the next rising clock edge it makes the latches transparent. By doing this *Wack* becomes equal to *Wreq* and *Rreq* becomes unequal to *Rack*, which results in making the latches opaque again. Since the cell waits until both conditions for a copy operation hold (being empty and write neighbour being full) before it synchronizes with the clock, it is universal in that it can be used for both write and read synchronization.

However, by including the synchronizing component in the feedback loop to the latch, we have extended the timing requirements with respect to the extended isochronic fork. Since its write neighbour is a free running cell, these extended timing requirements are not automatically fulfilled. One can weaken the timing requirements by speeding up the closing of the latch by means of an AND-gate after the UE4-component. But even in that case some additional delay in signal *Wack* is needed to fulfill the timing requirements.

The maximum frequency for a synchronizing buffer based on the universal cell is 581 MHz, which is only marginally faster than the maximum frequency of the read edge-synchronizing buffer, but 20% faster than the write edge-synchronizing buffer. Compared to the write edge-synchronizing buffer, a buffer based on universal cells has one disadvantage: the latency of an empty two-place synchronizing buffer is one clock tick.

5. Concluding remarks

We presented several designs of a fast buffer that can be used to bridge clock domains in GALS systems. The designs are based on two well-known ideas: pipeline synchronization and mousetrap buffers. We first combined both ideas and then in two steps improved the design by increasing its performance. Before doing the transformations we gave an elaborate analysis on the effect that delay asymmetries have on the performance of the design. Combin-



(a) Design

$$
\begin{aligned}
&(\ [(Rreq = Rack) * (Wreq \neq Wack)]; \ d\uparrow \\
&; \ [\overline{Clk}]; \ [Clk]; \ e\uparrow \\
&; \ (Rdat := Wdat \parallel Wack := Wreq \parallel Rreq := -Rack) \\
&; \ d\downarrow; \ e\downarrow \\
&)^*
\end{aligned}
$$

(b) Behaviour

Figure 11. Universal synchronization cell

ing both ideas resulted in a design with clock phase synchronization. In the first transformation we replaced phase synchronization by edge synchronization and in the second transformation we incorporated the edge synchronization circuit in the mousetrap cell. The resulting cell is a universal cell that can be used for both the read and the write side of the buffer.

If we compare the final design with the pipeline synchronization presented in [11] the following differences can be noticed.

- The handshake synchronization is based on clock edges instead of clock phases, which reduces the amount of synchronization circuitry. Consequently, the synchronizing overhead is less, which means a higher maximum clock frequency or for the same clock frequency a smaller probability of synchronization failure. An additional advantage is that the timing of the external events in relation to rising clock edges is well-defined, which is very important when we add circuitry offering the clocked interface.

- The design of the two-phase synchronizer (called symmetric synchronizer in [11]) uses one four-phase synchronizer instead of two.

- The similarity between the control circuitry of a mousetrap buffer cell and the circuitry transforming a four-phase synchronizer into a two-phase one, allowed us to incorporate the synchronization in the mousetrap cell.

Since the first transformation is independent of the design of the self-timed buffer cell, it can be applied in any pipeline synchronization design. The second transformation, however, is restricted to pipeline synchronization in mousetrap buffers.

Design	Write side	Read side
Phase	316	403
Edge	478	588
Universal	581	581

Table 1. Maximum clock frequencies (Mhz)

Table 1 shows –for the different designs– the maximum clock frequencies of both the write and the read side of the buffer.

Acknowledgement We thank Francesco Pessolano and Paul Wielage for drawing our attention to the mousetrap buffer.

References

[1] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, Oct. 1984.

[2] T. Chelcea and S. M. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proc. ACM/IEEE Design Automation Conference*, June 2001.

[3] M. d. Clercq and R. Negulescu. 1.1-GDI/s transmission between pausible clock domains. In *Proc. International Symposium on Circuits and Systems*, 2002.

[4] W. S. Coates, J. K. Lexau, I. W. Jones, S. M. Fairbanks, and I. E. Sutherland. FLEETzero: An asynchronous switch fabric chip experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 173–182. IEEE Computer Society Press, Mar. 2001.

[5] W. J. Dally and J. W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[6] J. Kessels, A. Peeters, P. Wielage, and S.-J. Kim. Clock synchronization through handshake signalling. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 59–68, Apr. 2002.

[7] A. J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[8] J. Muttersbach. *Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems*. PhD thesis, ETH, Zürich, 2001.

[9] M. Pechoucek. Anomalous response times of input synchronizers. *IEEE Transactions on Computers*, 25(2):133–139, Feb. 1976.

[10] C. L. Seitz. System timing. In C. A. Mead and L. A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[11] J. N. Seizovic. Pipeline synchronization. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 87–96, Nov. 1994.

[12] M. Singh and S. M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. In *Proc. International Conf. Computer Design (ICCD)*, pages 9–17, Nov. 2001.

[13] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[14] K. Y. Yun and A. E. Dooply. Pausible clocking-based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–488, Dec. 1999.

ID612437: Claims

1. Dataprocessing system comprising a first and second processing module which are mutually coupled by a synchronizing buffer, the buffer comprising a chain of pipeline elements, a pipeline element comprising

- a first latch for receiving an input data signal (Wdat) and providing a latched input data signal (Rdat), the latch having a transparent state wherein the latched input data signal is equal to the input data signal, and an opaque state wherein the latched input data signal retains its value,
- a controller for the first latch comprising
  - a second latch for receiving a first control signal (Wreq) indicative for the validity of the input data signal (Wdat), and for providing a latched first control signal (Rreq), the latch having a transparent state wherein the latched first control signal is equal to the first control signal, and an opaque state wherein the latched first control signal retains its value,
  - a comparator for comparing the latched first control signal (Rreq) with a second control signal (Rack) indicative whether the latched input data signal (Rdat) has been read, and for providing a latch control signal for controlling the state of the latches,

the pipeline comprising at least one synchronization element which synchronizes a control signal with a clock signal provided by one of the processing modules.

2. Dataprocessing system according to claim 1, wherein the synchronization element synchronizes one of the control signals with a phase of the clock signal.

3. Dataprocessing system according to claim 1, wherein the synchronization element synchronizes one of the control signals with a transition of the clock signal.

4. Dataprocessing system according to claim 1, 2 or 3, wherein the synchronization element synchronizes the second control signal (Rack) with the clock signal.

5. Dataprocessing system according to claim 4, wherein the pipeline comprises at least a first and a second pipeline element and wherein the synchronization element provides the second control signal (Rack) for the first pipeline element by synchronizing the latched first control signal (Rreq) of the second pipeline element with a clock signal provided by the first processing module (Wclk). (Figure 6, 10)

6. Dataprocessing system according to claim 1, 2 or 3, wherein the synchronization element synchronizes the first control signal (Wreq) with the clock signal.

7. Dataprocessing system according to claim 6, wherein the pipeline comprises at least a first and a second pipeline element and wherein the synchronization element provides the first control signal (Wreq) for the second pipeline element by synchronizing the latched first control signal (Rreq) of the first pipeline element with a clock signal provided by the second processing module (Rclk). (Figure 7, 10)

8. Dataprocessing system according to claim 3, wherein the controller further comprises a further comparator for comparing the first control signal (Wreq) with the latched first control signal (Rreq), the controller being arranged for providing the latch control signal (d) in response to first control signal (Wreq), the latched first control signal (Rreq) and the second control signal (Rack), and wherein the synchronization element provides the latch control signal (e) to the latches by synchronizing the latch control signal (d) provided by the comparator with the clock signal (Clk). (Figure 11)

9. Dataprocessing system according to claim 1, comprises a comparison element, a four-phase wait component and a latch, wherein the latch receives a request signal (Req) and provides the latched request signal as an acknowledge signal (Ack), the comparison element comparing the request signal (Req) and the acknowledge signal (Ack) and provides an input signal to the four-phase wait component, the four-phase wait component synchronizing this signal with the clock signal and providing a control signal to the latch.